University of Bahrain
College of Information Technology
Department of Computer Science
Summer Semester, 2011-2012
ITCS215 (Data Structures)

Mid Term Exam

Date: 30/07/2012 Time: 08:30 - 10:00

STUDENT NAME	
STUDENT ID #	DRAGON
SECTION #	

NOTE: THERE ARE SEVEN (7) PAGES IN THIS TEST
ONLY ONE SOLUTION WILL BE CONSIDERED FOR EACH QUESTION

QUESTION #	MARKS		COMMENTS
1	12	12	
2	10	10	
3	10	9.5	
4	10	10	
5	8	8	
TOTAL	50	49.5	

Question 1 [12 Marks]

Consider the following class definition:

```
class Item
{
    private:
      String name;
      long itemNum;

    public:
      Item( );
      Item(String itemName, long num);
     void setName(String itemName);
      void setItemNum(long num);
      string getName(  );
      long getItemNum( );
      void print( ); // prints name and itemNum
} // end Item
```

(A) Write a class called **StoreItem,** which inherits the properties of class **Item,** with inheritance type as public. This new class will have the following additional members:

Data members (private): price(double), quantity (int)

Member functions (public):

- set and get methods for both data members,
- print method to print all attributes (including that of **Item**),
- default constructor (without parameters)
- constructor with 4 parameters.

Write only prototypes of all member functions in the class **StoreItem**.

```
class StoreItem:public Item{
 private:
 double price;
 int quantity;
public:
Void  setprice(double p);
Void  setQuantity(int q);
double getprice();
int  getQuantity();
Void print();
StoreItem();
StoreItem(string n,long i,double p,int q);
};
```

✓ ✓ ✓

6

(B) Write definitions (implementation) of the following member functions of class **StoreItem**: `constructors` (both default and with parameters), and `print`.

```
// default conctructor
  StoreItem::StoreItem():Item() {
  price = 0.0;
 quantity = 0;
 }
  // constructor with parameters
StoreItem::StoreItem(string n,long i,double p,int q):Item(n, i) {
   price=p;
  quantity=q;
 }

// print
Void  StoreItem::print() {
   Item::print();
 cout<<"price :"<< price<<endl;
cout<<"Quantity:"<<quantity <<endl;
}
```

✓ ✓ ✓

6

Question 2 [10 Marks]

Write a function (not a member function) called **subSet** that accepts two objects ***L1*** and ***L2*** of type **arrayListType** as parameters. If all the elements of list ***L1*** are also in list ***L2*** (in any order), then the function returns true, else it returns false. If the list ***L1*** is empty, the function returns true. If the list ***L1*** is not empty but ***L2*** is empty, then the function returns false.

Function Prototype:

bool subSet(const arrayListType<Type>& *L1* ,const arrayListType<Type>& *L2*);

Assume that the class **arrayListType** is available for use.

Example 1:

L1: 10 5 7 8 2

L2: 15 **2** 12 **8** **5** 11 **10** 3 **7** 6

In this case, the function will return **true** as all the elements of L1 are in L2.

Example 2:

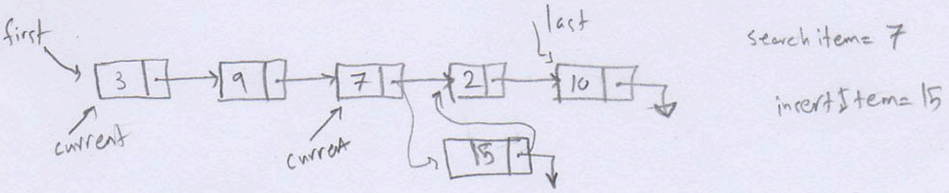
L1: 10 5 7 8 2

L2: 15 12 **8** **5** 11 **10** 3 6

In this case, the function will return **false**, as 2 and 7 are not there in L2.

```
template<class Type>
bool subSet(const arrayListType<Type>& L1, const arrayListType<Type>& L2){
if (L1.isEmpty())
return true;
if(L2.isEmpty())
return false;
Type item;
for(int i=0; i<L1.listSize();i++){
  L1.retrieveAt(i,item);
  if(L2.seqSearch(item)==-1)
    return false;
  }
return true;
}
```

10 ✓



Question 3 [10 Marks]

Write a function ***insertItem*** to be included as a member function in class **linkedListType**, which accepts an `insertItem` of type **Type** as a parameter and inserts it after the node containing element `searchItem` of type **Type** as info. Note that `searchItem` is also passed as a parameter to the function. If there is no node in the list containing element `searchItem` then insert `insertItem` at the end of the list. If the list is empty then create a linked list of one node having `insertItem` as the info.

Function prototype:

void insertItem(const Type& insertItem, const Type& searchItem);

Do not call any member function of class linkedlistType in your member function.

```
template<class Type>
Void linkedListType<Type>::insertItem(const Type& insertItem, const Type& searchItem){
 nodeType<Type> *newNode, * current;
 NewNode = new nodeType<Type>;
 assert(newNode != NULL);
 newNode->info = insertItem;
 newNode->link = NULL;
  count ++ ;
 if (first == NULL){
  first = newNode;
  last = newNode; }
 else{
 bool found = false;
 current = first;
 while(current != NULL && !found){
   if(current->info == searchItem)
      found = true;
   else
    current = current->link;
 }
   if (found){
    if (current->link == NULL){
     current->link = newNode;
     last = newNode; }
    else {
     current->link = newNode;
    newNode->link = current->link;
    }
   }
   else { // if not found
    last->link = newNode;
    last = newNode;
   }
  }
 }
```

9.5

Question 4 [10 Marks]

Write a member function called **degreeSorted** to be included in class **doublyLinkedList**, that returns the number of nodes that are sorted in ascending order divided by the total number of nodes in a doubly linked list. A node is defined as a "sorted node", if the value of its info is greater than the info of its previous node and less than the info of its next node. First node is a "sorted node", if its info is less than the info of the next node and last node is a "sorted node", if its info is greater than the info of its previous node. If the list is empty or has only one node, then the function returns 1.

The function prototype is:

```
double degreeSorted( );
```

Example:

list: **5** 10 8 **12** **30** 35 4 **50** **55** **60**

The "sorted nodes" in the above list are nodes having info 5, 12, 30, 50, 55 and 60. So, the number of sorted nodes = **6** and total number of nodes = **10**. Therefore, degree sorted = 6/10 = **0.6**. So, the function will return **0.6**.

```
template<class Type>
double doublyLinkedList<Type>::degreeSorted(){
 if(count<=1)
   return 1;
 int counter=0;
 nodeType<Type> *current=first;
 while(current != NULL){
 if( current== first && current->info <current->next->info)
      counter++;
 else if(current== last && current->info> current->back->info)
      counter++;
 else if ( current->info > current->back->info && current->info < current->next->info )
      Counter ++;
  current= current->next;
 }
 return (counter/(count*1.0));   OR
            (static_cast <double> counter) / count
 }
```

✓

10

Question 5 [8 Marks]

What is the output of the following program:

L1: 1 2 3 5 8

L2: ~~1~~ ~~2~~ ~~2~~ ~~3~~ ~~3~~ ~~5~~ ~~5~~ ~~8~~ ~~8~~ ~~13~~ 13 21

```
#include <iostream>
#include "arrayListType.h"
#include "linkedListType.h"
 using namespace std;

int     main( )
{
    arrayListType<int> L1(10);
    linkedListType<int> L2;
    int i, a, b, c;
    L2.insertLast(1);
    L2.insertLast(2);
    for (i= 0; i < 5; i++) {
        a = L2.front(); ~~1~~ ~~2~~ ~~3~~ ~~5~~ 8
        L2.deleteNode(a);
        b = L2.front(); ~~2~~ ~~3~~ ~~5~~ ~~8~~ 13
        L2.deleteNode(b);
        L1.insertEnd(a);
        L2.insertLast(b);
        L2.insertLast(a + b);
    }
    cout<<"Output: ";
    for (i= 0; i < 5; i++) {
        L1.retrieveAt(i, c);
        cout << c << "  ";
    }
    return 0;
}
```

~~0~~ ~~1~~ ~~2~~ ~~3~~ 4

Output: 1 2 3 5 8 ✓

8